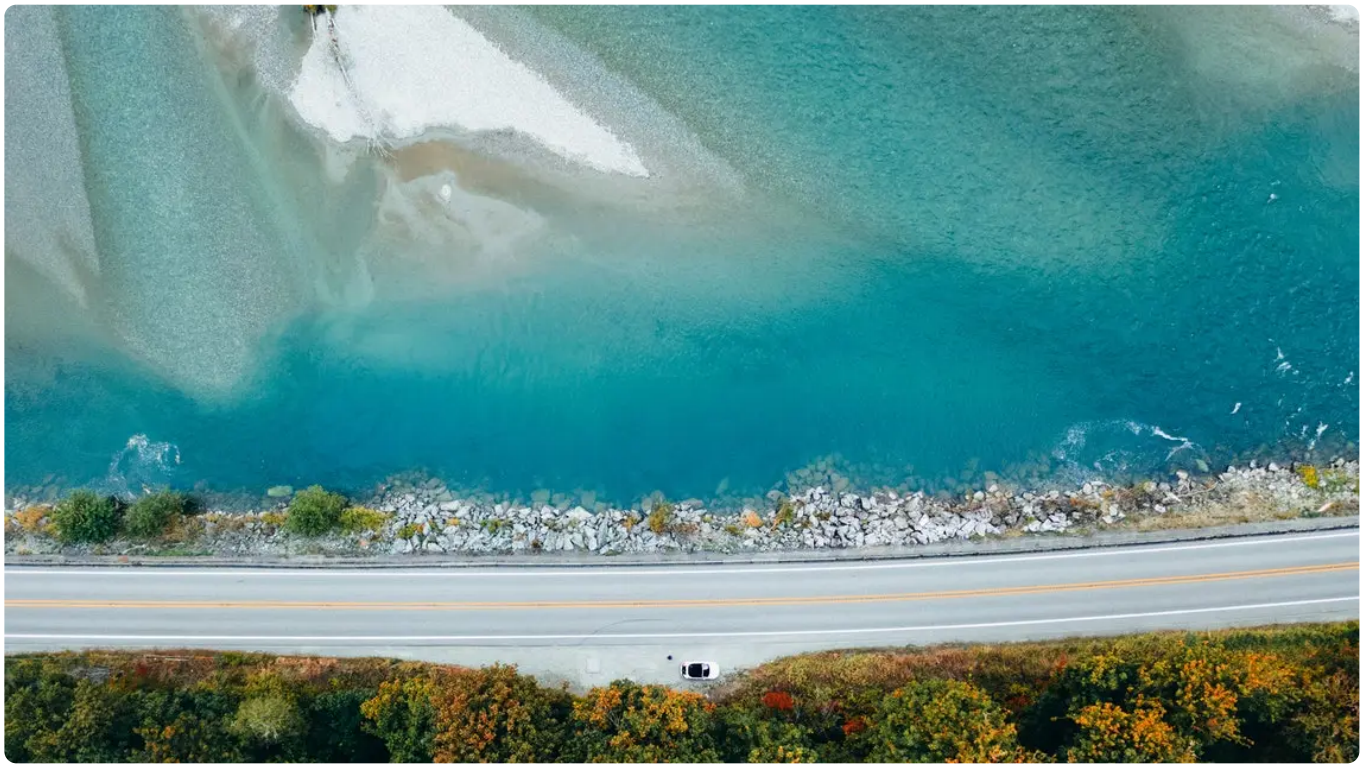
下载APP

02 | 文字块：怎么编写所见即所得的字符串？

2021-11-17 范学雷

《深入剖析Java新特性》 课程介绍 >



讲述：范学雷

时长 14:14 大小 13.05M

你好，我是范学雷。今天，我们聊一聊 Java 的文字块（text blocks）。

文字块这个特性，首先在 JDK 13 中以预览版的形式发布。在 JDK 14 中，改进的文字块再次以预览版的形式发布。最后，文字块在 JDK 15 正式发布。

文字块的概念很简单，它是一个由多行文字构成的字符串。既然是字符串，为什么还需要文字块这个新概念呢？文字块和字符串又有什么区别呢？我们还是通过案例和代码，来弄清楚这些问题吧。

阅读案例

我们在编写代码的时候，总是或多或少地要和字符串打交道。有些字符串很简单，比如我们都知道的"Hello，World！"字符串。有些字符串很复杂，里面可能有换行、对齐、转义字符、占位符、连接符等。

比如下面的例子中，我们要构造一个简单的表示"Hello，World！"的 HTML 字符串，就需要处理好文本对齐、换行字符、连接符以及双引号的转义字符。这就使得这段代码既不美观、也不简约，一点都不自然。

复制代码

```
String stringBlock =
        "<!DOCTYPE html>\n" +
        "<html>\n" +
        "    <body>\n" +
        "        <h1>\"Hello World!\"</h1>\n" +
        "    </body>\n" +
        "</html>\n";
```

这样的字符串不好写，不好看，也不好读。更糟糕的是，我们有时候需要从别的地方拷贝一段 HTML 或者 SQL 语句，然后再转换成类似于上面的字符串。是不是出力多，收效少，需要特别的耐心？遗憾的是，这样的工作还特别多，HTML, SQL, XML, JSON, HTTP, 随便就可以列一大堆。

不论对于写代码的人，还是阅读代码的人来说，处理这样的字符串都不是一件赏心悦目的事情。软件的质量是一个反馈系统，糟糕的事情总是可以让事情变得更糟糕。摊开来说，这样的字符串编写起来不省心，不仅消耗了更多时间，代码质量也没有保障。与此同时，复杂的语句也容易分散评审者的精力，让疏漏和错误不易被发现。

费时费力、质量还难以控制，这让复杂字符串的处理变成了一个很没有效率的事情。没有效率，也就意味着投入产出比低，所以我们就更不愿意投入精力和时间来做好这件事情。对于用户来说，糟糕的结果也会耗费他们更多的精力和时间。用户有多少，这个糟糕的成本就放大多少倍。

如果你经常需要阅读调试日志，你可能会有更深刻的体会。难以阅读的调试日志，可能会让你产生短暂的抗拒心理，甚至暂时地放弃调试，直到你的耐心又回来了。遗憾的是，提高调试日志的可读性，似乎永远排不上开发者的日程表。

这不是一个让人愉快的事情。不过，我们似乎也不曾有过更好的办法。

所见即所得的文字块

文字块是人们在试图扭转这种糟糕局面的过程中一个最重要的尝试。文字块是一个由多行文字构成的字符串。既然是字符串，文字块能有什么影响呢？其实，文字块是使用一个新的形式，而不是传统的形式，来表达字符串的。通过这个新的形式，文字块尝试消除换行、连接符、转义字符的影响，使得文字对齐和必要的占位符更加清晰，从而简化多行文字字符串的表达。

下面的这段代码，就是我使用文字块对阅读案例所做的改进。

复制代码

```
String textBlock = """
        <!DOCTYPE html>
        <html>
            <body>
                <h1>"Hello World!"</h1>
            </body>
        </html>
        """;
System.out.println(
        "Here is the text block:\n" + textBlock);
```

对比一下阅读案例里的代码，我们可以看到，下面的这些特殊的字符从这个表达式里消失了：

1. 换行字符（\n）没有出现在文字块里；
2. 连接字符（+）没有出现在文字块里；
3. 双引号没有使用转义字符（\）。

另外，出现在文字块开始和结束位置的，是三个双引号序列；而不是我们在字符串声明里看到的单个双引号。 **文字块由零个或多个内容字符组成，从开始分隔符开始，到结束分隔符结束。开始分隔符是由三个双引号字符（“”“）后面跟着的零个或多个空格，以及行结束符组成的序列。结束分隔符是一个由三个双引号字符（”“”）组成的序列。**

需要注意的是，开始分隔符必须单独成行；三个双引号字符后面的空格和换行符都属于开始分隔符。所以，一个文字块至少有两行代码。即使是一个空字符，结束分隔符也不能和开始分隔符放在同一行代码里。

复制代码

```
jshell> String s = """""";
|  Error:
|  illegal text block open delimiter sequence, missing line terminator
|  String s = """""";

jshell> String s = """
   ...> """;
s ==> ""
```

同样需要注意的是，**结束分隔符只有一个由三个双引号字符组成的序列。结束分隔符之前的字符，包括换行符，都属于文字块的有效内容。**

复制代码

```
jshell> String s = """
   ...> OneLine""";
s ==> "OneLine"


jshell> String s = """
   ...> TwoLines
   ...> """;
s ==> "TwoLines\n"
```

由于文字块不再需要特殊字符、开始分隔符和结束分隔符这些格式安排，我们几乎就可以直接拷贝、粘贴看到的文字，而不再需要特殊的处理了。同样地，你在代码里看到的文字块是什么样子，它实际要表达的文字就是什么样子的。这也就是说，“所见即所得”。

很多系统里常见的“所见即所得”的境界，终于也能够在 Java 语言里呈现出来了。

文字块的编译过程

那么，我们用文字块改进过的阅读案例，打印结果是什么样子的呢？从下面的打印结果，我们可以看到，为了代码整洁而使用的缩进空格并没有出现在打印的结果里。

复制代码

```
Here is the text block:
<!DOCTYPE html>
<html>
    <body>
        <h1>"Hello World!"</h1>
    </body>
</html>
```

也就是说，文字块的内容并没有计入缩进空格。文字块是怎么处理缩进空格的呢？这是我们学习文字块必须要了解的一个问题。

像传统的字符串一样，文字块是字符串的一种常量表达式。**不同于传统字符串的是，在编译期，文字块要顺序通过如下三个不同的编译步骤：**

1. **为了降低不同平台间换行符的表达差异，编译器把文字内容里的换行符统一转换成 LF（\u000A）；**
2. **为了能够处理 Java 源代码里的缩进空格，要删除所有文字内容行和结束分隔符共享的前导空格，以及所有文字内容行的尾部空格；**
3. **最后处理转义字符，这样开发人员编写的转义序列就不会在第一步和第二步被修改或删除。**

首先，我们从整体上来理解一下文字块的编译期处理这种方式。阅读一下下面的代码，你能不能预测一下下面这两个问题的结果？使用传统方式声明的字符串和使用文字块声明的字符串的内容是一样的吗？这两个字符串变量指向的是同一个对象，还是不同的对象？

复制代码

```
package co.ivi.jus.text.modern;

public class TextBlocks {
    public static void main(String[] args) {
        String stringBlock =
                "<!DOCTYPE html>\n" +
                "<html>\n" +
                "    <body>\n" +
                "        <h1>\"Hello World!\"</h1>\n" +
                "    </body>\n" +
                "</html>\n";

```

```
        String textBlock = """
                <!DOCTYPE html>
                <html>
                    <body>
                        <h1>"Hello World!"</h1>
                    </body>
                </html>
                """;

        System.out.println(
                "Does the text block equal to the regular string? " +
                stringBlock.equals(textBlock));
        System.out.println(
                "Does the text block refer to the regular string? " +
                (stringBlock == textBlock));
    }
}
```

第一个问题的答案应该没有意外，第二个问题的答案可能就会有意外出现了。使用传统方式声明的字符串和使用文字块声明的字符串，它们的内容是一样的，而且指向的是同一个对象。

该怎么理解这样的结果呢？其实，这就说明了，文字块是在编译期处理的，并且在编译期被转换成了常量字符串，然后就被当作常规的字符串了。所以，如果文字块代表的内容，和传统字符串代表的内容一样，那么这两个常量字符串变量就指向同一内存地址，代表同一个对象。

虽然表达形式不同，但是文字块就是字符串。既然是字符串，就能够使用字符串支持的各种 API 和操作方法。比如，传统的字符串表现形式和文字块的表现形式可以混合使用：

复制代码

```
System.out.println("Here is the text block:\n" +
        """
        <!DOCTYPE html>
        <html>
            <body>
                <h1>"Hello World!"</h1>
            </body>
        </html>
        """);
```

再比如，文字块可以调用字符串 String 的 API:

复制代码

```
int stringSize = """
         <!DOCTYPE html>
         <html>
             <body>
                 <h1>"Hello World!"</h1>
             </body>
         </html>
         """.length();
```

或者，使用嵌入式的表达式：

复制代码

```
String greetingHtml = """
         <!DOCTYPE html>
         <html>
             <body>
                 <h1>%s</h1>
             </body>
         </html>
         """.formatted("Hello World!");
```

巧妙的结束分隔符

好的，我们现在看看文字块编译的细分步骤。第一个和第二个步骤都很好理解。不过，第二个步骤里“删除共享的前导空格”，是一个我们可以巧妙使用的规则。通过合理地安排共享的前导空格，我们可以实现文字的编排和缩进。

为了方便理解，在下面的例子里，我们使用小数点号‘.’表示编译期要删除的前导空格，使用叹号‘!’表示编译期要删除的尾部空格。

第一个例子，我们把结束分隔符单独放在一行，和文本内容左边对齐。这时候，共享的前导空格就是文本内容本身共享的前导空格；结束分隔符仅仅是用来结束文字块的。这个例子里，我还加入了文字内容行的尾部空格，它们在编译期会被删除掉。

复制代码

```
// There are 8 leading white spaces in common
String textBlock = """
........<!DOCTYPE html>
........<html>
```

```
5 ........    <body>
6 ........        <h1>"Hello World!"</h1>!!!!
7 ........    </body>
8 ........</html>
9 ........""";
```

第二个例子，我们也把结束分隔符单独放在一行，但是放在比文本内容更靠左的位置。这时候，结束分隔符除了用来结束文字块之外，还参与界定共享的前导空格。

复制代码

```
1 // There are 4 leading white spaces in common
2 String textBlock = """
3 ....    <!DOCTYPE html>
4 ....    <html>
5 ....        <body>
6 ....            <h1>"Hello World!"</h1>!!!!
7 ....        </body>
8 ....    </html>
9 ...."""; 
```

第三个例子，我们也把结束分隔符单独放在了一行，但是放在文本内容左对齐位置的右侧。这时候，结束分隔符的左侧，除了共享的前导空格之外，还有多余的空格。这些多余的空格，就成了文字内容行的尾部空格，它们在编译期会被删除掉。

复制代码

```
1 // There are 8 leading white spaces in common
2 String textBlock = """
3 ........<!DOCTYPE html>
4 ........<html>
5 ........    <body>
6 ........        <h1>"Hello World!"</h1>!!!!
7 ........    </body>
8 ........</html>
9 ........!!!!""";
```

尾部空格还能回来吗？

你可能会问， 一般情况下，尾部空格确实没有什么实质性的作用。但是万一需要尾部空格，它们还能回来吗？

其实是可以的。为了能够支持尾部附带的空格，文字块还引入了另外一个新的转义字符，‘\s’，空格转义符。空格转义符表示一个空格。我们前面说过的文字块的编译器处理顺序，空格转义符不会在文字块的编译期被删除，因此空格转义符之前的空格也能被保留。所以，每一行使用一个空格转义符也就足够了。

下面的代码，就是一个重新带回尾部空格的例子，这个字符串的前两行就包含有尾部空格。

复制代码

```
// There are 8 leading white spaces in common
String textBlock = """
........<!DOCTYPE html>    \s!!!!
........<html>             \s
........    <body>!!!!!!!!!!
........        <h1>"Hello World!"</h1>
........    </body>
........</html>
........""";
```

该怎么表达长段落？

但是所见即所得的文字块也有一个小烦恼。我们知道，编码规范一般都限定每一行的字节数，通常是 80 个或者 120 个字节。可是一个文本的长段落通常要超出这个限制。文字块里的换行符通常需要保留，编码规范通常要遵守，那该如何表达长段落或者长行呢？

针对这种情况，文字块引入了一个新的转义字符，‘< 行终止符 >’，换行转义符。换行转义符的意思是，如果转移符号出现在一个行的结束位置，这一行的换行符就会被取缔。下面的例子就使用了换行转义符，它就把分散在两行的"Hello World!"连接在一行里了。

复制代码

```
String textBlock = """
        <!DOCTYPE html>
        <html>
            <body>
                <h1>"Hello \
        World!"</h1>
            </body>
        </html>
        """;
```

需要注意的是，上面的例子里，换行转义符之前，还有一个空格。这个空格会被删除吗？连接后的字符，是没有空格间隔的“HelloWorld!”，还是中间有空格的“Hello World!”？还记得我们前面说过的编译器处理顺序吗？空格处理先于转义字符处理。因此，换行转义符之前的空格不算是文字块的尾部空格，因此会得到保留。

总结

好，到这里，今天的课程就要结束了，我来做个小结。从前面的讨论中，我们了解了文字块的基本概念，它的表达形式以及编译的过程。

文字块是 Java 语言中一种新的文字。 字符串能够出现的任何地方，也都可以用文字块表示。但是，文字块提供了更好的表现力和更少的复杂性。 文字块“所见即所得”的表现形式，使得使用复杂字符串的代码更加清晰，便于编辑，也便于阅读。这是一个能够降低代码错误，提高生产效率的改进。

如果要丰富你的代码评审清单，学习完这一节内容后，你可以加入下面这一条：

> 复杂的字符串，使用文字块表述是不是更清晰？

另外，通过今天的讨论，我拎出了几个技术要点，这些都可能在你的面试中出现。通过这一次学习，你应该能够：

知道文字块的基本概念，以及文字块和字符串的关系；

- 面试问题：你知道 Java 的文字块吗？它和字符串有什么区别？

了解文字块要解决的问题，并且能够准确使用文字块；

- 面试问题：应当什么时候使用文字块？

了解文字块的表达形式，编译过程以及文字块特有的转义字符。

- 面试问题：怎么用文字块实现文本缩进？

如果能够有意识地使用文字块，你应该能够大幅度提高复杂字符串的可读性。从而更快地编写代码，也让潜在的错误更少。毫无疑问，在面试的时候，有意识地在代码里使用文字块，除了节省时间之外，还能够让你的代码更容易阅读和接受，给面试官带来新鲜的感受。

思考题

在前面的讨论里，我们说过文字块是一个“所见即所得”的字符串表现形式。我们可以直接拷贝、粘贴文字段落到代码里，而不需要大量的调整。可是，在有些场景里，要想完全地实现“所见即所得”，仅仅使用文字块，可能还是要费一点周折的。

比如说吧，我们看到的诗，有的时候是页面居中对齐的。比如下面的这首小诗，采用的格式就是居中对齐。

No man is an island,

Entire of itself,

Every man is a piece of the continent,

A part of the main.

居中对齐这种形式，在 HTML 或者文档的世界里，很容易处理，设置一下格式就可以了。如果是用 Java 语言，该怎么处理好这首小诗的居中对齐问题？这就是今天我们的思考题。

稍微提示一个，你可以使用添加缩进空格的方式对齐，也可以不局限于简单的、单纯的 Java 语言，比如添加进来 HTML 的文本。

欢迎你在留言区留言、讨论，分享你的阅读体验以及你对这个思考题的想法。

注：本文使用的完整的代码可以从GitHub下载，你可以通过修改GitHub上review template代码，完成这次的思考题。如果你想要分享你的修改或者想听听评审的意见，请提交一个 GitHub 的拉取请求（Pull Request），并把拉取请求的地址贴到留言里。这一小节的拉取请求代码，请放在实例匹配专用的代码评审目录下，建一个以你的名字命名的子目录，代码放到你专有的子目录里。比如，我的代码，就放在 text/review/xuelei 的目录下面。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | JShell：怎么快速验证简单的小问题？

下一篇 03 | 档案类：怎么精简地表达不可变数据？

精选留言 (13)

写留言

旺仔double
2021-11-17

本篇主要讲了几点：
1. 认识 textarea 是什么？基本语法是什么？
2. textarea 和 原始的长字符串处理方式产生的对象指向同一内存地址。
3.文字块可以调用字符串 String 的 API 。
4.可以使用嵌入式的表达式 。 ...
展开

作者回复: 建议置顶！

可惜我不知道有没有置顶的功能。 如果点赞可以帮助置顶的话，小伙伴们都来点赞吧。总结的太好了！

共 2 条评论 > 12

森林
2021-11-17

老师，java啥时候支持字符串插值。
展开

作者回复: 这个特性，我还不知道是不是有计划。 现在有替换的方案，虽然没有字符串差值简单，所以可能不是很紧急。 如果觉得这个好，可以去OpenJDK提议，推动它的早日支持。

1

Geek_f022bc
2021-11-17

什么时候能支持字符串插值啊

展开

作者回复: 出现了两次，看来这个需求很旺盛啊。你想去OpenJDK提交一个新特性的申请吗？

共 2 条评论 > 1

aoe
2021-11-17

还是Go中的``用起来方便啊。""" """确实也方便了很多，但是平时用的真不多，复杂的文字一般都会从模板中读取/展示，例如：MyBatis的XML文件、Yaml文件、FreeMarker、Thymeleaf等

作者回复: 嗯，场景不同，使用方式差别很大。 从模版中读取是一个值得小伙伴们学习的好办法。

我自己的代码很多要记日志，这个特性算是解放了我可怜的眼睛。

共 3 条评论 > 1

旺仔double
2021-11-17

直接添加缩进空格的实现
String poem = """
No man is an island,
entire of itself,
every man is a piece of the continent,...

展开

作者回复: 很遗憾极客时间的评论不能准确地显示源代码。时间允许的话，你可以试一试专栏的GitHub的PR请求。每一讲的末尾，都有一个注解，提醒大家怎么获得源代码，怎么提交PR请求。

共 2 条评论 > 1

ABC
2021-11-18

感觉Python3里面的文本格式化和js里面的模板字符串更方便一点,用Java的文字块插值的时

候要记住顺序,稍微繁琐了一些.希望后续版本文字块能参照Python3或者js改进一下.

作者回复: 是的，字符串插值看起来很有吸引力。已经有好几个人在留言去提到了。建议写信给jdk-dev@openjdk.java.net，提议这个新特性。有人提要求，优先级才能上得去。

Jxin
2021-11-18

没有对比就没有伤害。相对于以前，相对于其他语言。

作者回复: 哈哈

will
2021-11-18

老师formatted是会创建一个新的对象吗？

展开

作者回复: 问题字面看不出来你会怎么使用formatted。 你使用JShell试一试？

共 2 条评论 >

飞翔
2021-11-18

java11有这个特性嘛

展开

作者回复: “文字块这个特性，首先在 JDK 13 中以预览版的形式发布。在 JDK 14 中，改进的文字块再次以预览版的形式发布。最后，文字块在 JDK 15 正式发布。”所以，JDK 11不支持文字块。

Calvin
2021-11-18

老师，看完这节，我有点疑问，想请教一下：

1：编译过程去除共享的前导空格，这个可以理解是为了处理代码缩进，但是为什么文本行后面的空格也要设计为去除呢？这样设计的初衷是什么呀？（这样设计，为了保留后面的空格还要使用\s转义，感觉反而更麻烦了！）...

展开

作者回复: 1). 通常状况下，行末的空格都是不必要，毕竟看不到。这样设计，省去了肉眼判断行末是否有空格的麻烦。 毕竟，有空格肉眼不好看，而转义字符就清晰很多了。 多数情况下，行末的转义字符应该用不上。

2），3），5）的答案，建议你试一试。

4）使用转义字符，\"""。6）继续使用转义字符。这和普通的字符串的转义字符的用法是一致的。

PS：有可能是词语选择的偏好导致的。“不换行转义符”和“换行转义符”似乎都有可能导致理解偏差。 小伙伴们，也帮我想一想有没有更好的表达方式。

共 3 条评论

注定非凡

2021-11-17

这个新特性和Scala的文字块好像啊

展开

作者回复: 毕竟Scala也要跑在JVM上，所以关键字（"""）的选择空间不多；而且，语言的设计都是尽量靠近，降低开发者切换的学习曲线。

bigben

2021-11-17

有人缩进用tab怎么办？每个人的ide的缩进空格数量设置的也不一样，是不是也会导致问题？

作者回复: 我想是有问题的， 你有没有试试看？ tab的问题，需要编码规范来约束。

共 3 条评论

3.141516

2021-11-17

JShell 算是借鉴脚本语言提供一个 playground，Swift 在诞生初期就具备了这个特性。

本节的文字块印象中也是 JavaScript 有的特性。

作者回复: 嗯， 也许编程语言的终局，就是殊途同归。